Discrete: Lab I

Kareem Tarek Ibrahim

20011112

Abdullah Mohamed Abdullah

20010906

I. Problem Statement

Q1. Basic Bit Operations

You have to implement 4 bits operations, so your programe might allow user choose one of the
following operations.

1. getBit(int number, int position):This function returns the bit value (an integer, 0 or 1) in the number at position position, according to its binary representation. The least significant bit in a number is position 0.

2. setBit(int number, int position):This function set the bit value (to be 1) in the
number at position position, according to its binary representation. The least significant
bit in a number is position 0 and return number after setting the bit.

3. clearBit(int number, int position):This function cleat the bit value (to be 0) in the
number at position position, according to its binary representation. The least significant bit in a number is position 0 and return number after clearing the bit.

4. updateBit(int number, int position, boolean value):This function set the bit value
according to value parameter(an integer, 0 (false) or 1(true) in the number at position position, according to its binary representation. The least significant bit in a number is position 0 and return number after update.

Q2. Sets Operations using Bits manipulation
Write a program that takes
1. An input a list of strings as a Universe
2. Then takes another input a number of sets (that are subsets of the universe)
3. Then ask the user about the operations they want to perform (3 required features to be implemented in this assignment):
   (a) Union of two sets.
   (b) Intersection of two sets
   (c) Complement of a set
4. You must use bits to represent set and use bits manipulation to implement this
part, you can use the following video https://www.youtube.com/watch?v=qZ59R0EQg8w to understand the solution.

Q3. Applications for bits manipulation
1. Given a non-empty array of integers nums, every element appears twice except for one.
Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space. you must think for your solution using bits manipulation operation.
2. Write a function that takes an unsigned integer and returns the number of ’1’ bits it.

II. Used data strucutures

We didn't use any data structure in the first and the third questions. We used a map in the second question.

We used to represent every string in the universe with a number that presents an index. This index will be used, where ith index represents the ith string. It will be zero or one depending on the existence of the string in the sets.

For example if the universe is: {“AS”, “D

F”, “GH”, “JK”, “L”, “ZX”, “CV”, “BN”, “M,”}

The map will be like.

AS	DF	GH	JK	L	ZX	CV	BN	M,
0	1	2	3	4	5	6	7	8

Table1. An instance of the used map.

III. Algorithm with pseudo codes

Q1. We depended on bitwise manipulation operations, mainly the left shift of zero, with other operations. For the first part we used and with it, second with or, and third we used not operation with left shift.

The fourth part depended on the second and third parts.

```
int getBit(int number,int pos)
{
    int x=((1<<pos)&number)
    if(x!=0)
        x=1
    return x
}

int setBit(int number,int pos)
{
    int x=((1<<pos)|number)
    return x
}

int clearBit(int number,int pos)
{
    int x=((~(1<<pos))&number)
    return x
}

int updateBit(int number,int pos,bool
value)
{
    int x
    if(value)
        x=setbit(number,pos)
    else
        x= clearbit(number,pos)
    return x
}
```

Q2. We used Hashing algorithms, which depended on a map to give a number to each string. Then we used a 2d array with rows presenting the sets and ith column presenting the ith string, if it's 1 this means that this set in that row includes this string, if zero this means it isn't in this set.

By this move, we were able to use bits manipulation operations, as every set consists of zeros and ones

	AS	DF	GH	JK	L	ZX	CV	BN
Set1	1	0	0	1	0	1	0	1
Set2	0	0	0	0	0	1	1	0
Set3	1	1	1	0	0	1	0	1
Set4	1	1	1	1	1	1	1	1
Set5	0	0	1	0	1	1	0	0
Set6	1	0	1	0	1	0	1	0
Set7	0	0	1	1	0	0	0	0

Table2. Arbitrary values and sets

The pseudocode of this part:

```
string x
cin >> x
dict1[x] = i
dict2[i] = x
```

Where dict1 is a string key map, and dict2 is an integer key map.

Then records the hashing table is presented in this way:

```
cin >> numberOfSits
    for(long long i = 0 i < numberOfSits
i++)
    {
        cin >> setSize
         for(long long j = 0 j < setSize
j++)
        {
            string buffer
            cin >> buffer
            hashing[i][dict1[buffer]] = 1
        }
    }
```

Then we can do the union with ore, intersection with and, and complement by getting the elements with zero. We return to the map once more with the index we are in now if the result of the bitwise operation is one to know the string that has this key.
This part Is large so please check the code.

```
void unions()
{
    cout << "Please insert the number of
first set you want to union" << endl
    long long first
    cin >> first
    cout << "Please insert the number of
second set you want to union" << endl
    long long second
    cin >> second
    first--, second--
    bool tmam = false
    cout << "{"
    long long counterforprinting = 0
    for(int i = 0 i < numberOfElements i++)
    {
        long long x = hashing[first][i] |
hashing[second][i]
        if(x == 1)
        {
            counterforprinting++
            if(counterforprinting != 1)
                cout << ", "
            cout << dict2[i]
            tmam = true
        }
    }
    cout << "}" << endl

    if(!tmam)
```

```
        cout << "Inserted sets doesn't
exist" << endl
    cout << endl
}

void intersect()
{
    cout << "Please insert the number of
first set you want to intersect" << endl
    long long first
    cin >> first
    cout << "Please insert the number of
second set you want to intersect" << endl
    long long second
    cin >> second
    first--, second--
    ok tmam = false
    cout << "{"
    long long counterforprinting = 0
    for(int i = 0 i < numberOfElements i++)
    {
        long long x = hashing[first][i] &
hashing[second][i]
        if(x == 1)
        {
            counterforprinting++
            if(counterforprinting != 1)
                cout << ", "
            cout << dict2[i]
            tmam = true
        }
    }
    cout << "}" << endl
    if(!tmam)
        cout << "Inserted sets doesn't
exist" << endl
    cout << endl
}
void complement()
{
    cout << "Please insert the number of
set you want to complement" << endl
    long long first
    cin >> first
    first--
    ok tmam = false
    cout << "{"
    long long counterforprinting = 0
    for(int i = 0 i < numberOfElements i++)
    {
        long long x = 1-hashing[first][i]
        if(x == 1)
        {
            counterforprinting++
            if(counterforprinting != 1)
                cout << ", "
            cout << dict2[i]
            tmam = true
        }
    }
    cout << "}" << endl
    if(!tmam)
        cout << "Inserted sets doesn't
exist" << endl
    cout << endl
}
```

Q3. In the first part we use the xor operation to remove the repetitions

.

```
for (int i = 0; i <  n; ++i) {
        cin>>a[i]
    }

    for (int i = 0; i < n; i++) {
        number=number^a[i]
    }
```

We loop until the number equals zero. Get the right bit in the `number` and check whether it's one or not, if it is, we increase the counter by one. If not we decrease the value. And after that we right shift.

```
while (number!=0){

        if((number&(1))!=0){
            count++;
        }
        number=number>>1;

    }
```

IV. Code Snippets

On Github.

V. Sample Runs

On Github.

VI. Assumptions

We assumed that the number of sits will not exceed 1000, and the number of elements in each set won't exceed 1000 also.
We believed that increasing the size of the hashing array above 1 million is dangerous, so we took this decision.